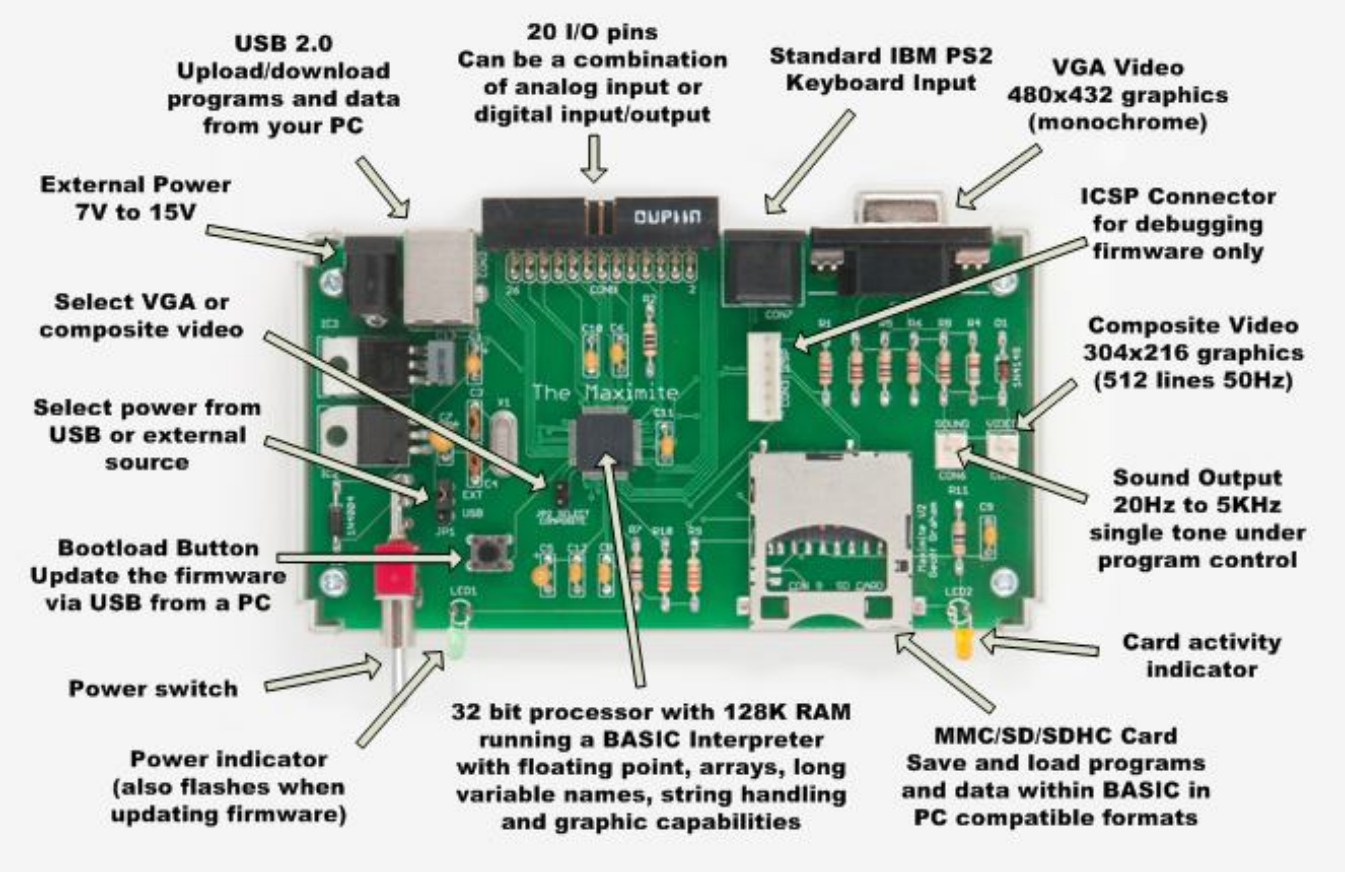
Maximite User Manual

For updates to this manual go to http://geoffg.net/maximite.html.


USB 2.0
Upload/download programs and data from your PC
20 I/O pins
Can be a combination of analog input or digital input/output
Standard IBM PS2 Keyboard Input
VGA Video
480x432 graphics (monochrome)
External Power
7V to 15V
ICSP Connector for debugging firmware only
Select VGA or composite video
Composite Video
304x216 graphics
(512 lines 50Hz)
Select power from USB or external source
Sound Output
20Hz to 5KHz single tone under program control
Bootload Button
Update the firmware via USB from a PC
Card activity indicator
Power switch
32 bit processor with 128K RAM running a BASIC Interpreter with floating point, arrays, long variable names, string handling and graphic capabilities
MMC/SD/SDHC Card
Save and load programs and data within BASIC in PC compatible formats
Power indicator
(also flashes when updating firmware)

Video Output

Placing a jumper on JP2 will select composite video output (eg, a TV set), removing it will select VGA video output (only one can be plugged in at a time).

VGA

Standard monochrome VGA (31.5KHz horizontal scanning with 60Hz vertical refresh).

480x432 pixel graphic screen. 80 characters per line and 36 lines per screen

Composite

Standard monochrome PAL (15.625KHz horizontal scanning with 50Hz vertical refresh non interlaced).

304x216 pixel graphic screen. 50 characters per line and 18 lines per screen

USB

Implements the CDC (Communication Device Class) protocol over USB 2.0. This is a serial interface to the BASIC interpreter so, by using a terminal emulator on the host, programs can be entered, edited and run. Using this interface you can upload programs or download (using the BASIC LIST command).

The Windows driver is available from http://geoffg.net/maximite.html. There is native support for the CDC protocol in Linux (the cdc-acm driver) and Apple OS/X.

Keyboard

Standard IBM compatible PS2 keyboard with mini-DIN connector or a USB/mini-DIN adapter.

Non ASCII keys (such as the function keys) are mapped to ASCII characters. Use a command like `PRINT HEX$(VAL(INKEY$))` to check the actual mapping.

SD/MMC Card Interface

Will accept MMC, SD or SDHC memory cards formatted as FAT16 or FAT32. File names must be in 8.3 format (long file names are not supported). Note that there is no advantage in using a fast SD card as the card is clocked at a fixed 20MHz, regardless of its speed rating.

Electrical Characteristics

Power Supply

Via External Power:	7V to 12V (14V if no significant current is drawn from the I/O pins). The centre pin of the external power connector is positive.
Via USB Connector:	4.5V to 5.5V (JP1 placed in the USB position)
Current Draw:	140mA typical (plus current draw from the I/O pins)

Digital Inputs

Logic Low:	0 to 0.65V
Logic High:	2.5V to 3.3V (I/O pins 1 to 10) 2.5V to 5.5V (I/O pins 11 to 20)
Input Impedance:	>1MΩ. All digital inputs are Schmitt Trigger buffered.
Frequency Response:	Up to 200KHz (pulse width 10nS or more) on the counting inputs (pins 11 to 14).

Analog Inputs (I/O pins 1 to 10)

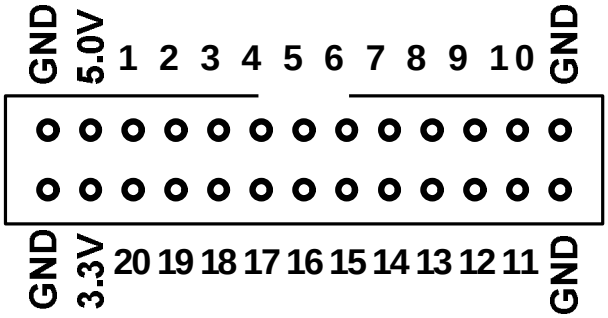
Voltage Range:	0 to 3.3V
Accuracy:	Typically better than ±1% although this can be considerably improved by using a correction factor in the BASIC program.
Input Impedance:	>1MΩ (for accurate readings the source impedance should be <10K)

Digital Outputs

Typical current draw or sink ability on any I/O pin:	10mA
Maximum current draw or sink on any I/O pin:	25mA
Maximum current draw or sink for all I/O pins combined:	150mA
Maximum open collector voltage (I/O pins 11 to 20):	5.5V

External Input/Output Connector

Rear panel connector with the pin numbers as used in MMBasic (external view looking at the back panel):



Loading New Firmware

Maximite includes the ability to reprogram itself with a new version of its firmware – also known as re-flashing or a boot loading. To do this, hold down the PROGRAM button on the PC board while you apply power and the front panel power LED will flash to indicate that the Maximite is in the reprogramming mode. Firmware upgrades can be downloaded from http://geoffg.net/maximite.html and will come with a program called "BootLoader.exe", run this program and follow the instructions included in the upgrade package to reprogram the Maximite with the new version.

Maximite BASIC V2.1

Functional Summary

Keyboard/Display

Input can come from either a keyboard or from a computer using a terminal emulator via the USB interface. Both the keyboard and the USB interface can be used simultaneously and can be detached or attached at any time without affecting a running program.

Output will be simultaneously sent to the USB interface and the video display (VGA or composite). Either can be attached or removed at any time.

Startup

On startup MMBasic looks for a file called "AUTORUN.BAS" in the root directory of the SD card and will automatically load and run it if found.

If it is not found MMBasic will print a prompt (">") and wait for input.

Command and Program Input

At the prompt you can enter a command line followed by the enter key and it will be immediately run. This is useful for testing commands and their effects. If the line is preceded with a number it will be saved in memory along with other lines in the program. The program can be listed with LIST and run using the RUN command.

You can interrupt MMBasic at any time by typing CTRL C on either the keyboard or USB interface and control will be returned to the prompt.

A program line can be changed by entering a new line with the same number. A line can be deleted by entering its number without any commands. All program lines can be cleared from working memory with the NEW command.

Multiple commands separated by a colon can be entered on the one line (as in INPUT A : PRINT B).

SD Card Storage

A program can be saved to the SD card using the SAVE command. It can be reloaded using LOAD or merged with the current program using MERGE. A saved program can also be loaded and run using the RUN command. The RUN command can also be used within a running program which enables one program to load and transfer control to another.

Data files can be opened using OPEN and read from using INPUT, LINE INPUT or INPUT$() or written to using PRINT or WRITE. Both data and programs are stored using standard text and can be read and edited in Windows, Apple Mac, Linux, etc.

You can list the programs stored on the SD card with FILES, delete them using KILL and rename them using NAME. The current working directory can be changed using CHDIR. A new directory can be created with MKDIR or an old one deleted with RMDIR.

Whenever specified a file name can be a string constant (ie, enclosed in double quotes) or a string variable. This means you must use double quotes if you are directly specifying a file name. Eg, RUN "TEST.BAS"

Expressions

In most cases where a number or string is required you can also use an expression. For example:

FNAME$ = "TEST": RUN FNAME$ + ".BAS"

Or, as an extreme (and not recommended) example:

NBR = 100 : GOTO NBR * 3 + 20

Structured Statements

MMBasic supports a number of modern structured statements. The IF… THEN command can span may lines with ELSEIF … THEN, ELSE and ENDIF statements as required and also spaced over many lines. The DO WHILE … LOOP command and its variants make it easy to build loops without using the GOTO statement.

Timing

You can get the current date/time using the DATE and TIME functions and you can set them by assigning the new date and time to them. If not set the calendar will start from midnight 1st Jan 2010 on power up.

You can freeze program execution for a number of milliseconds using PAUSE. MMBasic also maintains an internal stopwatch function (the TIMER function) which counts up in milliseconds. You can is reset TIMER to zero or any other number by using TIMER as a command.

Using SETTICK you can setup a "tick" which will generate a regular interrupt with a period from one millisecond to over a month. See Interrupts below.

External Input/Output

You can configure an external I/O pin using the SETPIN command, set its output using the PIN()= command and read the current input value using the PIN() function. Digital I/O uses the number zero to represent a low voltage and any non zero number for a high voltage. An analogue input will report the measured voltage as a floating point number.

Interrupts

Any external I/O pin can be configured to generate an interrupt using the SETPIN command with up to 21 interrupts (including the tick interrupt) active at any one time. Interrupts can be setup to occur on a rising or falling digital input signal and will cause an immediate branch to a specified line number (similar to a GOSUB). The target line number can be the same or different for each interrupt. Return from an interrupt is via the IRETURN statement. All statements (including GOSUB/RETURN) can be used within an interrupt.

If two or more interrupts occur at the same time they will be processed in order of pin numbers (ie, an interrupt on pin 1 will have the highest priority). During processing of an interrupt all other interrupts are disabled until the interrupt routine returns with an IRETURN. During an interrupt (and at all times) the value of the interrupt pin can be accessed using the PIN() function.

A periodic interrupt (or regular "tick") with a period specified in milliseconds can be setup using the SETTICK statement. This interrupt has the lowest priority.

Interrupts can occur at any time but they are disabled during INPUT statements. If you need to get input from the keyboard while still accepting interrupts you should use the INKEY$ function. When using interrupts the main program is completely unaffected by the interrupt activity unless the interrupts alters variables used by the main program.

For most programs Maximite will respond to an interrupt in under 100µS. To prevent slowing the main program by too much an interrupt should be short and execute the IRETURN statement as soon as possible. Also remember to disable an interrupt when you have finished needing it – background interrupts can cause strange and non intuitive bugs.

Graphics

Graphics commands operate on the video output. Coordinates are measured in pixels with the top left of the screen at location x = 0 and y = 0.

You can clear the screen with CLS and an individual pixel can be turned on with PSET or turned it off with PRESET. You can draw lines and boxes with LINE, and circles using CIRCLE. You can also set the screen location (in pixels) of the next PRINT output using LOCATE.

Sound

The SOUND command will generate a simple square wave between 20Hz and 5KHz lasting for a specified duration.

Compatibility

MMBasic implements a large subset of Microsoft's GW-BASIC. There are numerous small differences due to physical and practical considerations but most MMBasic commands and functions are essentially the same. An online manual for GW-BASIC is available at http://www.xs4all.nl/~hwiegman/gw-man and this provides a more detailed description of the commands and functions that are available.

MMBasic also implements a number of modern programming structures documented in the ANSI Standard for Full BASIC (X3.113-1987). These include the DO WHILE … LOOP and structured IF .. THEN … ELSE … ENDIF statements.

Operators and Precedence

The following operators, in order of precedence, are recognised. Operators that are on the same level (for example + and -) are processed with a left to right precedence as they occur on the program line.

Arithmetic operators:

^	Exponentiation
* / \ MOD	Multiplication, division, integer division and modulus (remainder)
+ -	Addition and subtraction

Logical operators:

NOT	logical inverse of the value on the right
<> < > <= => >= >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	equality
AND OR XOR	Conjunction, disjunction, exclusive or

The operators AND, OR and XOR are bitwise operators. For example PRINT 3 AND 6 will output 2.

The other logical operations result in the number 0 (zero) for false and 1 for true. For example the statement PRINT 4 >= 5 will print the number zero on the output and the expression A = 3 > 2 will store +1 in A.

The NOT operator is highest in precedence so it will bind tightly to the next value. For normal use the expression to be negated should be placed in brackets. For example, IF NOT (A = 3 OR A = 8) THEN …

String operators:

+	Join two strings
<> < > <= => >= >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	equality

Naming Conventions

Command names, function names, variable names, file names, etc are not case sensitive, so that "Run" and "RUN" are equivalent and "dOO" and "Doo" refer to the same variable.

There are two types of variable; numeric which stores a floating point number (eg, 45.386) and string which stores a string of characters (eg, “Tom”). String variables are terminated with a $ symbol (eg, name$) while numeric variables are not.

Variable names can start with an alphabetic character or underscore and can contain any alphabetic or numeric character, the period (.) and the underscore (_). They may be up to 32 characters long. A variable name must not be the same as a command, function or keyword. Eg, data = 5 is illegal as DATA is a command.

Constants

Numerical constants may begin with a numeric digit (0-9) for a decimal constant, &H for a hexadecimal constant, &O for an octal constant or &B for a binary constant. For example &B1000 is the same as the decimal constant 8.

Decimal constants may be preceded with a minus (-) or plus (+) and may terminated with 'E' followed by an exponent number to denote exponential notation. For example 1.6E+4 is the same as 16000.

String constants are surrounded by double quite (“) marks. Eg, “Hello World”.

Implementation Characteristics

Maximum length of a command line is 255 characters.

Maximum length of a variable name is 32 characters.

Maximum number of dimensions to an array is 8.

Maximum number of arguments to commands that accept a variable number of arguments is 50.

Numbers are stored and manipulated as single precision floating point numbers. The maximum number that can be represented is 3.40282347e+38 and the minimum is 1.17549435e-38

The range of integers (whole numbers) than can be manipulated without loss of accuracy is ±16777100.

Maximum string length is 255 characters.

Maximum number of files simultaneously open is 10.

Maximum SD card size is 2GB formatted with FAT16 or 2TB formatted with FAT32.

Predefined Variables

MM.HRES	The horizontal resolution of the current video display screen in pixels.
MM.VRES	The vertical resolution of the current video display screen in pixels.
MM.ERRNO	Is set to the error number if a statement involving the SD card fails or zero if the operation succeeds. This is dependent on the setting of OPTION ERROR. The possible values for MM.ERRNO are: 0 = No error 1 = No SD card found 2 = SD card is write protected 3 = No space on the card 4 = All root directory entries are taken 5 = Invalid filename 6 = Cannot find file 7 = Cannot find directory 8 = File is read only 9 = Cannot open file 10 = Error reading from file 11 = Error writing to file 12 = Not a file 13 = Not a directory 15 = Directory not empty 15 = Unspecified error accessing the SD card

Commands

‘ (single quotation mark)	Starts a comment and any text following it will be ignored. Comments can be placed anywhere on a line.
CHDIR dir$	Change the current working directory on the SD card to ‘dir$’ The special entry “..” represents the parent of the current directory and “.” Represents the current directory.
CIRCLE (x, y) ,r [,c [,F]]	Draws a circle on the video output centred at x and y with a radius of r. If c is zero the pixels are turned off, if it is non zero or not specified the pixels are turned on (ie, the circle is drawn). The F option will cause the circle to be filled according to the c parameter. Note that because of differing aspect ratios in monitors the circle will be oval to some degree.

CLEAR	Will delete all variables and recover the memory used by them. See ERASE for deleting specific array variables.
CLOSE [#]nbr [,[#]nbr] …	Will close the files previously opened with the file number 'nbr'. The # is optional. Also see the OPEN command.
CLS	Clears the video display screen.
CONTINUE	Will resume running a program that has been stopped by an END statement, an error, or CTRL-C. The program will restart with the next statement following the previous stopping point.
DATA constant[,constant]...	Stores numerical and string constants to be accessed by READ. String constants do not need to be quoted unless they contain significant spaces, the comma or a keyword (such as GOSUB, DO, etc).
DATE$ = "DD-MM-YY" or DATE$ = "DD/MM/YY"	Set the date of the internal clock/calendar. DD, MM and YY are numbers, for example: DATE$ = "28-2-2011" The date is set to "1-1-2000" on power up.
DELETE line DELETE -lastline DELETE firstline - DELETE firstline - lastline	Deletes a program line or a range of lines. If –lastline is used it will start with the first line in the program. If startline- is used it will delete to the end of the program. Also see the NEW command.
DIM variable(elements...) [variable(elements...)]...	Specifies variables that have more than one element in a single dimension, i.e., arrayed variables.
DO <statements> LOOP	This structure simply loops; the only way out is by EXIT or GOTO.
DO WHILE expression <statements> LOOP	Loops while "expression" is true (this is equivalent to the older WHILE-WEND loop, also implemented in MMBasic).
DO <statements> LOOP UNTIL expression	Loops until the expression following UNTIL is true.
ELSE	Introduces a default condition in a multiline IF statement. See the multiline IF statement for more details.
ELSEIF expression THEN	Introduces a secondary condition in a multiline IF statement. See the multiline IF statement for more details.
ENDIF	Terminates a multiline IF statement. See the multiline IF statement for more details.
END	Will end the running program and return to the command prompt

ERASE variable [,variable]...	Deletes arrayed variables and frees up the memory. Use CLEAR to delete all variables including all arrayed variables.
ERROR [error_msg$]	Will force an error and terminate the program. This is normally used in debugging or to trap events that should not occur.
EXIT [FOR]	EXIT by itself exits from a DO...LOOP EXIT FOR exits from a FOR...NEXT loop.
FILES [search_pattern$]	List the program files in the current directory on the SD card. The optional 'search_pattern$' may contain question marks (?) to match any character. An asterisk (*) as the first character of the filename or extension will match any file or any extension. If omitted, all files will be listed.
FOR counter = start TO finish [STEP increment]	Initiates a FOR-NEXT loop with the 'counter' initially set to 'start' and incrementing in 'increment' steps (default is 1) until 'counter' equals 'finish'. The 'increment' must be an integer but may be negative.
GOSUB line	Initiates a subroutine call to the line specified. The subroutine must end with RETURN.
GOTO line	Branches program execution to the specified line.
IF expr THEN statement OR IF expr THEN statement ELSE statement	Evaluates the expression 'expr' and performs the THEN statement if it is true or skips to the next line if false. The optional ELSE statement is the reverse of the THEN test. The 'statement' can be just a line number and in that case a GOTO is assumed. For Microsoft compatibility the 'THEN statement' construct can be also replaced with 'GOTO linenumber'. This type of IF statement is all on one line.

IF expression THEN
<statements>
[ELSE
<statements>]
[ELSEIF expression THEN
<statements>]
ENDIF
	Multiline IF statement with optional ELSE and ELSEIF cases and ending with ENDIF. Each component is on a separate line.
	Evaluates 'expression' and performs the statement(s) following THEN if the expression is true or optionally the statement(s) following the ELSE statement if false.
	The ELSEIF statement (if present) is executed if the previous condition is false and it starts a new IF chain with further ELSE and/or ELSEIF statements as required.
	One ENDIF is used to terminate the multiline IF.

INPUT ["prompt string";] list of variables	Allows input from the keyboard to a list of variables. The input command will prompt with a question mark (?). The input must contain commas to separate each data item if there is more than one variable. For example, if the command is: INPUT a, b, c And the following is typed on the keyboard: 23, 87, 66 Then a = 23 and b = 87 and c = 66 If the "prompt string" is specified it will be printed before the question mark. If the prompt string is terminated with a comma (,) rather than the semicolon (;) the question mark will be suppressed.

INPUT #nbr, list of variables	Same as above except that the input is read from a file previously opened for INPUT as 'nbr'. See the OPEN command.
IRETURN	Will return from an interrupt. The next statement to be executed will be the one that was about to be executed when the interrupt was detected.
KILL file$	Deletes the program specified by file$ from the SD card. Quote marks are required around a string constant. Example: KILL "SAMPLE.DAT"
LET variable = expression	Assigns the value of 'expression' to the variable. LET is automatically assumed if a line does not start with a command.
LINE [(x1 , y1)] - (x2, y2) [,c [,B[F]]]	Draws a line or box on the video screen. x1,y1 and x2,y2 specify the beginning and end points of a line. c specifies the displayed pixel (0 = off, non zero = on) and defaults to on if not specified. (x1, y1) is optional and if omitted the last drawing point will be used. The optional B will draw a box with the points (x1,y1) and (x2,y2) at opposite corners. The optional BF will draw a box (as ,B) and fill the interior. x is the horizontal coordinate while y is the vertical. The top left hand corner is 0, 0.
LINE INPUT [prompt$,] string-variable$	Reads entire line from the keyboard or a file or device into 'string-variable$'. If specified the 'prompt$' will be printed first. Unlike INPUT, LINE INPUT will read a whole line, not stopping for comma delimited data items. A question mark is not printed unless it is part of 'prompt$'.
LINE INPUT #nbr, string-variable$	Same as above except that the input is read from a file previously opened for INPUT as 'nbr'. See the OPEN command.
LIST LIST line LIST -lastline LIST firstline - LIST firstline - lastline	Lists all lines in a program line or a range of lines. If –lastline is used it will start with the first line in the program. If startline- is used it will list to the end of the program.
LOAD file$	Loads a program called 'file$' from the SD card into working memory. Quote marks are required around a string constant. Example: LOAD "TEST.BAS"
LOCATE x, y	Positions the cursor to a location in pixels and the next PRINT command will place its output at this location. The top left corner is 0, 0. Only affects the video output.
LOOP [UNTIL expression]	Terminates a program loop: see DO.

MEMORY	List the amount of memory currently in use. For example:
	`5kB (17%) Program memory used`
	`3kB (16%) Variable memory used`
	`12kB (30%) Array and string memory used`
	Program memory is cleared by the NEW command. Variable, array and string memory spaces are cleared by many commands (eg, NEW, RUN, LOAD, etc) as well as the specific commands CLEAR and ERASE.

MERGE file-name	Adds program lines from 'file-name' to the program in memory. Unlike LOAD, it does not clear the program currently in memory.
MKDIR dir$	Make, or create, the directory ‘dir$’ on the SD card.
NAME old$ AS new$	Will rename a file or a directory on the SD card from ‘old$’ to ‘new$’
NEW	Deletes the program in memory and clears all variables.
NEXT [counter-variable]	NEXT comes at the end of a FOR-NEXT loop; see FOR.
ON variable GOTO|GOSUB line[,line,line,...]	ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of variable; if it is 1, the first line is called, if 2, the second line is called, etc.
OPEN fname$ FOR mode AS [#]fnbr	Will open a file on the SD card for reading or writing. ‘fname’ is the filename (8 chars max) with an optional extension (3 chars max) separated by a dot (.). ‘mode’ is INPUT or OUTPUT or APPEND. INPUT will open the file for reading and throw an error if the file does not exist. OUTPUT will open the file for writing and will automatically overwrite any existing file with the same name. APPEND will also open the file for writing but it will not overwrite an existing file, instead any writes will be appended to the end of the file. If there is no existing file the APPEND option will act the same as the OPEN mode (i.e. the file is created then opened for writing). ‘fnbr’ is the file number (1 to 10). The # is optional. Up to 10 files can be open simultaneously. The INPUT, LINE INPUT, PRINT, WRITE and CLOSE commands as well as the EOF() and INPUT$() functions all use ‘fnbr’ to identify the file being operated on. See also OPTION ERROR and MM.ERRNO for error handling.
OPTION BASE 0 or OPTION BASE 1	Sets the lowest value for array subscripts to either 0 or 1. The default is 0.
OPTION ERROR CONTINUE or OPTION ERROR ABORT	Sets the treatment for errors in file input/output. The option CONTINUE will cause MMBasic to ignore file related errors. The program must check the variable MM.ERRNO to determine if and what error has occurred. The option ABORT sets the normal behaviour (ie, stop the program and print an error message). The default is ABORT. Note that this option only relates to errors reading or writing from the SD card, it does not affect the handling of syntax and other program errors.

PAUSE nbr	Will halt execution of the running program for 'nbr' milliseconds. The maximum value of 'nbr' is 4294967295 (about 49 days).
PIN(pin) = value	For a 'pin' configured as digital output this will set the output to low ('value' is zero) or high ('value' non zero). You can set an output high or low before it is configured as an output and that setting will be the default output when the SETPIN command takes effect. 'pin' zero is a special case and will always control the LED on the front panel. A 'value' of non zero will turn the LED on, or zero for off. See the function PIN() for reading from a pin and the command SETPIN for configuring it.
PRINT expression [[,;]expression] … etc	Outputs text to the screen. Multiple expressions can be used and must be separated by either a: • Comma (,) which will output the tab character • Semicolon (;) which will not output anything and if present at the end of the line will suppress the automatic output of a carriage return/ newline at the end of a print statement When printed a number is preceded with a space if positive or a minus (-) if negative but is not followed by a space. Integers (whole numbers) are printed without a decimal point while fractions are printed with the decimal point and the significant decimal digits. Large numbers (greater than six digits) are printed in scientific format. The function FORMAT$() can be used to format numbers. The function TAB() can be used to space to a certain column and the string functions can be used to justify or otherwise format strings.
PRINT #nbr, expression [[,;]expression] … etc	Same as above except that the output is directed to a file previously opened for OUTPUT or APPEND as 'nbr'. See the OPEN command.
PRESET (x, y)	Turn off a pixel on the video screen. x is the horizontal coordinate while y is the vertical. The top left corner is 0, 0.
PSET (x, y)	Turn on a pixel on the video screen.
RANDOMIZE nbr	Seeds the random number generator with 'nbr'. To generate a different random sequence each time you must use a different value for 'nbr'. One good way to do this is use the TIMER function. For example 100 RANDOMIZE TIMER
READ variable[, variable]...	Reads values from DATA statements and assigns these values to the named variables. Variable types in a READ statement must match the data types in DATA statements as they are read. See also DATA and RESTORE.
REM string	REM allows remarks to be included in a program. Note the Microsoft style use of the single quotation mark to denote remarks is also supported and is preferred.
RESTORE	Resets the line and position counters for DATA and READ statements to the top of the program file.
RETURN	RETURN concludes a subroutine called by GOSUB.

RMDIR dir$	Remove, or delete, the directory 'dir$' on the SD card.
RUN [line] [file$]	Executes the program in memory. If a line number is supplied, then execution begins at that line. Or, if a file name (file$) is supplied, the current program will be erased and that program will be loaded from the SD card and executed. This enables one program to load and run another. Example: RUN "TEST.BAS"
SAVE [file$]	Saves the program in the current working directory on the SD card as 'file$'. Example: SAVE "TEST.BAS"
SETPIN pin, cfg [,line]	Will configure the external I/O 'pin' according to 'cfg': 0 Not configured or inactive 1 Analog input (pins 1 to 10) 2 Digital input (all pins and 5V tolerant on pins 11 to 20) 3 Frequency input (pins 11 to 14) 4 Period input (pins 11 to 14) 5 Counting input (pins 11 to 14) 6 Interrupt on low to high input change (all pins) 7 Interrupt on high to low input change (all pins) 8 Digital output (all pins) 9 Open collector digital output to 5V (pins 11 to 20) See the function PIN() for reading inputs and the statement PIN()= for outputs. If an interrupt is configured (ie, 'cfg' is 6 or 7) then the line number of the interrupt is specified in the third parameter 'line'. The value of the pin can always be retrieved using the function PIN(). See also IRETURN. The interrupt can be disabled by using SETPIN with 'cfg' not 6 or 7.
SETTICK period, line	This will setup a periodic interrupt (or "tick"). The time between interrupts is 'period' milliseconds and 'line' is the line number of the interrupt routine. See also IRETURN. The period can range from 1 to 4294967295 mSec (about 49 days). This interrupt can be disabled by setting 'line' to zero (ie, SETTICK 0, 0).
SOUND frequency, duration	Generate a single tone of 'frequency' (between 20Hz and 5KHz) for 'duration' milliseconds. This is a square wave which is played in the background and does not stop program execution. If 'duration' is zero, any active SOUND statement is turned off. If no SOUND statement is running, a 'duration' of zero has no effect.
TIME$ = "HH:MM:SS" or TIME$ = "HH:MM" or TIME$ = "HH"	Sets the time of the internal clock. MM and SS are optional and will default to zero if not specified. For example TIME$ = "14:30" will set the clock to 14:30 with zero seconds. The time is set to "0:0:0" on power up.
TIMER = msec	Resets the timer to a number of milliseconds. Normally this is just used to reset the timer to zero but you can set it to any positive integer. See the TIMER function for more details.

TROFF	Turns of the trace facility; see TRON.
TRON	Turns on the trace facility. This facility will print each line number in square brackets as the program is executed. This is useful in debugging programs.
WEND	WEND concludes a WHILE-WEND loop; see WHILE.
WHILE expression	WHILE initiates a WHILE-WEND loop. The loop ends with WEND, and execution reiterates through the loop as long as the 'expression' is true. This construct is included for Microsoft compatibility. New programs should use the DO … LOOP construct.
WRITE [#nbr,] expression [,expression] …	Outputs the value of each ‘expression’ separated by commas (,). If the ‘expression’ is a number it is outputted without preceding or trailing spaces. If it is a string it is surrounded by double quotes (“). The list is terminated with a new line. If ‘#nbr’ is specified the output will be directed to a file on the SD card previously opened for OUTPUT or APPEND as ‘#nbr’. See the OPEN command. WRITE is useful for writing data that will later be read by the INPUT command or by programs that can read CSV (comma separated variables) format files (such as Microsoft’s Excel).

Functions

ABS(number)	Returns the absolute value of the argument 'number'.
ASC(string$)	Returns the ASCII code for the first letter in the argument ‘string$’.
ATN(number)	Returns the arctangent value of the argument 'number' in radians.
CHR$(number)	Returns a one-character string with the character corresponding to the ASCII code indicated by argument 'number'.
CINT(number)	Round numbers with fractional portions up or down to the next whole number or integer. For example, 45.47 will round to 45 45.57 will round to 46 -34.45 will round to -34 -34.55 will round to -35 See also INT() and FIX().
COS(number)	Returns the cosine of the argument 'number' in radians.
DATE$	Returns the current date based on MMBasic’s internal clock as a string in the form "DD-MM-YYYY". For example, “28-02-2010”. The internal clock/calendar will keep track of the time and date including leap years. The date is set to “1-1-2010” on power up. To set the time use the command DATE$.

EOF([#]nbr)	Will return true if the file previously opened for INPUT with the file number ‘nbr’ is positioned at the end of the file. The # is optional. Also see the OPEN, INPUT and LINE INPUT commands.
EXP(number)	Returns the exponential value of 'number'.
FIX(number)	Truncate a number to a whole number by eliminating the decimal point and all characters to the right of the decimal point. For example 9.89 will return 9 and -2.11 will return -2. The major difference between FIX and INT is that FIX provides a true integer function (ie, does not return the next lower number for negative numbers as INT() does). This behaviour is for Microsoft compatibility. See also CINT() .
FORMAT$(nbr [, fmt$])	Will return a string representing ‘nbr’ formatted according to the specifications in the string ‘fmt$’. The format specification starts with a % character and ends with a letter. Anything outside of this construct is copied to the output as is. The structure of a format specification is: % [flags] [width] [.precision] type Where ‘flags’ can be: - Left justify the value within a given field width 0 Use 0 for the pad character instead of space + Forces the + sign to be shown for positive numbers space Causes a positive value to display a space for the sign. Negative values still show the – sign ‘width’ is the minimum number of characters to output, less than this the number will be padded, more than this the width will be expanded. ‘precision’ specifies the number of fraction digits to generate with an e, or f type or the maximum number of significant digits to generate with a g type. If specified the precision must be preceded by a dot (.). ‘type’ can be one of: g Automatically format the number for the best presentation. f Format the number with the decimal point and following digits e Format the number in exponential format If uppercase G or F is used the exponential output will use an uppercase E. Examples: format$(45) will return 45 format$(45, “%g”) will return 45 format$(24.1, “%g”) will return 24.1 format$(24.1,”%f”) will return 24.1000 format$(24.1, “%e”) will return 2.41000e+01 format$(24.1,"%09.3f") will return 00024.100 format$(24.1,"%+.3f") will return +24.100 format$(24.1,"**%-9.3f**") will return **24.100 **
HEX$(number)	Returns a string giving the hexadecimal (base 16) value for the 'number'.

INKEY$	Reads the status of the keyboard, and returns a single character if available. If a character is not available, this function will immediately return with a null string ("").
INPUT$(nbr, [#]fnbr)	Will return a string composed of 'nbr' characters read from a file previously opened for INPUTwith the file number 'fnbr'. This function will read all characters including carriage return and new line without translation. The # is optional. Also see the OPEN command.
INSTR([start-position,] string-searched$, string-pattern$)	Returns the position at which string-pattern$ occurs in string-searched$, beginning at start-position.
INT(number)	Truncate an expression to the next whole number less than or equal to the argument. For example 9.89 will return 9 and -2.11 will return -3. This behaviour is for Microsoft compatibility, the FIX() function provides a true integer function. See also CINT() .
LEFT$(string$, number-of-spaces)	Returns a substring a string$ with number-of-spaces from the left (beginning) of the string).
LEN(string$)	Returns the number of characters in string$.
LOG(number)	Returns the natural logarithm of the argument 'number'.
LCASE(string$)	Returns 'string$' converted to lowercase characters.
MID$(string$, start-position-in-string[, number-of-chars])	Returns a substring of 'string$' beginning at 'start-position-in-string' and continuing for 'number-of-chars' bytes. If 'number-of-chars' is omitted the returned string will extend to the end of 'string$'
OCT$(number)	Returns a string giving the octal (base 8) representation of 'number'.
PIN(pin)	Returns the value on the external I/O 'pin'. Zero means digital low, 1 means digital high and for analogue inputs it will return the measured voltage as a floating point number. Frequency inputs will return the frequency in Hz (maximum 200KHz). A period input will return the period in milliseconds while a count input will return the count since reset (counting is done on the positive rising edge). The count input can be reset to zero by resetting the pin to counting input (even if it is already so configured). 'pin' zero is a special case which will always return the state of the bootload push button on the PC board (non zero means that the button is down). Also see the SETPIN and PIN() = commands.
POS	Returns the current cursor position in the line.
RIGHT$(string$, number-of-spaces)	Returns a substring a string$ with number-of-spaces from the right (end) of the string).

RND(number)	Returns a pseudo-random number. The 'number' value is ignored if supplied. The RANDOMIZE command reseeds the random-number generator.
SGN(number)	Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.
SIN(number)	Returns the sine of the argument 'number' in radians.
SPACE$(number)	Returns a string of blank spaces 'number' bytes long.
SPC(number)	Returns a string of blank spaces 'number' bytes long. This function is similar to the SPACE$() function and is only included for Microsoft compatibility.
SQR(number)	Returns the square root of the argument 'number'.
STR$(number)	Returns a string in the decimal (base 10) representation of the argument 'number'.
STRING$(number, ascii-value\|string$)	Returns a string 'number' bytes long consisting of either the first character of string$ or the character answering to the ASCII value ascii-value.
TAB(number)	Outputs spaces until the column indicated by 'number' has been reached.
TAN(number)	Returns the tangent of the argument 'number' in radians.
TIME$	Returns the current time based on MMBasic 's internal clock as a string in the form "HH:MM:SS" in 24 hour notation. For example, “14:30:00”. The internal clock/calendar will keep track of the time and date including leap years. The time is set to midnight on power up. To set the time use the command TIME$.
TIMER	Returns the elapsed time in milliseconds (eg, 1/1000 of a second) since reset. If not specifically reset this count will wrap around to zero after 49 days. The timer is reset to zero on power up and you can also reset it by using TIMER as a command. See the TIMER command.
UCASE(string$)	Returns ‘string$’ converted to uppercase characters.
VAL(string$)	Returns the numerical value of the ‘string$’.